Subsetting a Variable

This tutorial describes three different ways of subsetting a variable. The three methods subset the variable into an identical physical domain.

---

1. IOsubset

```
## import cdtime module for dates
import cdtime

## define start and end date
startCd = cdtime.componenttime(1996,6,24,12)
endCd = cdtime.componenttime(1996,6,24,22)

## Define a spatial domain in terms of
## Latitude and Longitude in degrees N and E
lat1 = 37.02
lat2 = 39.276
lon1 = -92.694
lon2 = -90.358

## Subset the variable
o3_sub1 = o3.IOsubset([(lon1,lat1),(lon2,lat2)], timeLst = [startCd,endCd])
```

The *IOsubset* method is the most flexible way to subset your data. You can use it to subset based on any of the three spatial coordinate systems: lon lat, xlon ylat, or row column. In addition, you can use *IOsubset* to make temporal slices using mx.DateTime object, cdtime objects, strings, or a time index, and you can make vertical slices with either sigma layers or layer index.

In this example, we first import the cdtime module so that we can create two cdtime dates. After creating the dates and defining the Latitude and Longitude in degrees, we then call the *IOsubset* method to subset the variable. In all of these examples, the subsetted variable is an iovar object and retains the metadata from the original variable.

. Contents

---

2. Call

```
## import cdtime module for dates
import cdtime

## define start and end date
startCd = cdtime.componenttime(1996,6,24,12)
endCd = cdtime.componenttime(1996,6,24,22)

## Define spatial domain interms of xLons and yLats
## i.e. meters from projection center
xlon1 = -234000.
xlon2 = -30000.
ylat1 = -318000.
ylat2 = -78000.
```

```
## subset the variable
o3_sub2 = o3(time=(startCd,endCd), \
        latitude=(ylat1,ylat2,"ccb"), \
        longitude=(xlon1,xlon2,"ccb"))
```

Here, we are making a call to the iovar object to subset the variable. Unlike *IOsubset*, you must define your spatial domain in terms of xLon and yLat (meters from the projection center). Your temporal domain can be defined in either cdtime objects or string format. You can use cdms selectors since this is identical to a cdms variable call (see the CDAT documentation for more specifics on selectors).

. Contents

---

3. Slice

```
## Use array slicing to subset the variabls
o3_sub3 = o3[6:17,:,5:26,3:21]

## Compare the 3 subsets
o3_sub1.getValue()[0,0,0]
o3_sub2.getValue()[0,0,0]
o3_sub3.getValue()[0,0,0]

## All 3 variables should give
[ 0.03051318, 0.03054752, 0.03044657,
        0.03035855, 0.03041357, 0.03033318,
        0.03067152, 0.03059888, 0.03069421,
        0.03070652, 0.03061113, 0.03060304,
        0.03050215, 0.0307868 , 0.03130329,
        0.0317276 , 0.03208612, 0.0333391 ,]
##

o3_sub1.getLatitude()
o3_sub2.getLatitude()
o3_sub3.getLatitude()
## All 3 variables should give
   id: yLat
   Designated a latitude axis.
   units:  meters
   Length: 21
   First:  -318000.0
   Last:   -78000.0
   Other axis attributes:
      long_name: y coordinate of projection
      standard_name: projection_y_coordinate
      axis: Y
##
```

Here, we are using array notation to subset the variable. In python, the slicing convention for arrays and lists is somewhat different than in other languages. The slice does not include the upper bounds and the index is 0 based. For example, "6:17" means array elements 6 through 16. In our example subset, we extract elements 6 through 17 from the first dimension (time), all the second dimension (layer), elements 5 through 25 from the third dimension (row), and 3 through 20 from the fourth dimension (column). See the Numerics documentation for more specifics on array slices.

After making our three subsets, we can make some cursory comparisons to show that they are identical.

In terms of computational efficience, slicing is the most efficient and *IOsubset* has the most overhead. In

terms of conceptual flexibility, *IOsubset* is the most flexible and slicing is the most restrictive.